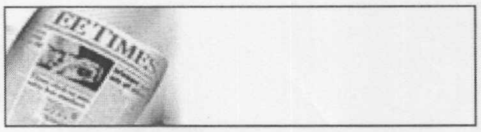

EETIMES NETWORK
LOGIN / REGISTER

SEA
Sear
Exa
Sear
LM3

EE Expert Robert Ashby

Embedded Engineering

SITE NAVIGATOR
Customize
Knowledge Centers ▸
Product Reviews
Data Sheets ▸
Guides & Experts ▸
News ▸
International ▸
Ask Us
Circuit Cellar
Online Tools
App Notes
NetSeminars
Careers
Resources ▸
FAQ
EE Times Network ▸
Electronics Group Sites ▸

Archive | Main EE Expert | Guides & Experts

Shifting Gears

by Robert Ashby

I wrote an article a few months ago about Simple Control where I described some simple equations that would work well for many control situations similar to a PID loop control. The system that I had envisioned while writing the article was a micro that would output a PWM signal sent to a DC motor to achieve a certain speed feedback. The other control situation that I've worked with consists of a switching power system that would send a constant voltage to a DC motor to achieve a certain position. This type of system can be controlled with the help of a very simple equation such as the one illustrated below.

```
FullOnCode:
   cp   desiredposition, actualposition
          ;Compare tells me where to move
   jz   stopmotor
          ;If they are the same, stop the motors
   jc   gobackward
          ;Go back if needed
goforward:           ;Otherwise go forward
   bset motorforward
          ;Turn motor on going forward
   bclr motorbackward
          ;Turn motor off going backward
   jmp  endmotor   ;exit
gobackward:
   bclr motorforward
          ;Turn motor off going forward
   bset motorbackward
          ;Turn motor on going backward
   jmp  endmotor   ;exit
stopmotor:
   bclr motorforward
          ;All stop on motors
   bclr motorbackward
endmotor:
```

This routine seems quite simple and straightforward, but it can run into some difficulties if the sampling rate controlling the motor is slow. You can overshoot your target and end up with oscillations. Even if the sample time is quick enough, you might need to consider is the response time of the motor system. If the system continues to move past another position after you have told it to stop, you can also stop with beyond the mark, try to move back, and continuously overshoot. In either situation, you end up with oscillations.

One solution is to downshift by implementing a poor-man's PWM. By varying a PWM of adequate frequency sent to the DC motor that I've described above, you can effectively slow down the motor and thereby suffice with a slow response system or a slower sampling rate. Shifting gears is easier than it first might seem. The easiest task to accomplish is to gear down by half. This changes the above code only slightly. The changes are shown in red.

```
GearedDownByHalf:
   cp   desiredposition, actualposition
         ;Compare tells me where to move
   jz   stopmotor
         ;If they are the same, stop the motors
   jc   gobackward
         ;Go back if needed
goforward:           ;Otherwise go forward
   bnot motorforward
         ;Toggle motor on going forward
   bclr motorbackward
         ;Turn motor off going backward
   jmp  endmotor   ;exit
gobackward:
   bclr motorforward
         ;Turn motor off going forward
   bnot motorbackward
         ;Toggle motor on going backward
   jmp  endmotor   ;exit
stopmotor:
   bclr motorforward
         ;All stop on motors
   bclr motorbackward
endmotor:
```

Note: If you don't have a `bit not` (`bnot`) macro or instruction, then use the `xor` statement on the port register. Here's an example.

```
xor   PORT0, 0x02      ;Toggle bit 1 of PORT0
```

Slower gear ratios can be achieved easily by using an extra register. This carries a great flexibility advantage that allows you to slow down to various different speeds by changing only the gear divisor. You can accomplish this by adding code similar to the following before the `FullOnCode` listed earlier.

```
VariableGear:
   sub  gear,#1
         ;Increment towards a carry condition
   jnc  stopmotor
         ;Stop motors most of the time
   ld   gear,#3
         ;This is my gear divisor
FullOnCode:
```

The gear divisor of 3 results in a speed 1/3 that of the original code. A divisor of 4 results in a speed ¼ of the original code, and so on. As you lower the speed of the motor using the methods that I've described above, note that lower speeds result in lower PWM frequencies, i.e., a gear divisor of 6 has twice the period of a gear divisor of 3.

If you want to maintain the higher speed of the motor in moving long distances, then you can check to see how close you are to your target before implementing the `VariableGear` code.

```
SpeedCheck:
   mov  A,desiredposition
   sub  A,actualposition
   jnc  SpeedCheck1
         ;Continue if result was positive
   xor  A,0xFF
```

```
            ;Otherwise make A positive
   Inc  A
SpeedCheck1:
   cmp  A,#HIGHGEAR
            ;If closer than HIGHGEAR
   jc   SpeedCheck2
            ;Continue with divide
   mov  gear,#0
            ;Otherwise go full speed
SpeedCheck2:
VariableGear:
```

All that remains is to test your system to see when you need to slow down to determine the HIGHGEAR constant and then determine how slow your gear divider needs to be to make your position-feedback system a working reality.

Embedded Engineering Archive

Guides and Experts Analog Avenue EDA Tools PLD DSP EDA Embedded Systems Power Test

E-mail This Article   Printer-Friendly Page

About ChipCenter-Questlink Contact Us Privacy Statement Advertising Information FAQ